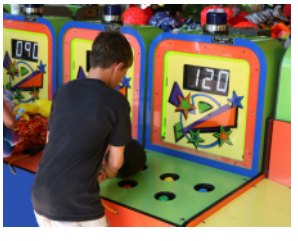
CHAPTER 3

MoleMash



This chapter shows you how to create MoleMash, a game inspired by the arcade classic Whac-A-Mole™, in which mechanical critters pop out of holes, and players score points when they successfully whack them with a mallet. MoleMash was created by a member of the App Inventor team, ostensibly to test the sprite functionality (which she implemented), but really because she's a fan of the game.

When Ellen Spertus joined the App Inventor team at Google, she was eager to add support for creating games, so she volunteered to implement sprites. The term, originally coined to describe mythological creatures such as fairies and pixies, was adopted by the computing community in the 1970s to refer to images capable of movement on a computer screen (for video games). Ellen first worked with sprites when she attended a computer camp in the early 1980s and programmed a TI 99/4. Her work on sprites and MoleMash was motivated by double nostalgia—for both the computers and games of her childhood.

What You'll Build

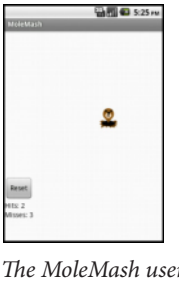


Figure 3-1. The MoleMash user interface

For the MoleMash app shown in Figure 3-1, you'll implement the following functionality:

- A mole pops up at random locations on the screen, moving once every second.
- Tapping the mole causes the device to vibrate, increment a display of "hits" (increasing it by one), and move the mole immediately to a new location.
- Tapping the screen but missing the mole increments a display of "misses."
- Pressing a Reset button resets the counts of hits and misses.

What You'll Learn

The tutorial covers the following components and concepts:

- The `ImageSprite` component for touch-sensitive movable images.
- The `Canvas` component, which acts as a surface on which to place the `ImageSprite`.
- The `Clock` component to move the sprite around once every second.
- The `Sound` component to produce a vibration when the mole is tapped.
- The `Button` component to start a new game.
- Procedures to implement repeated behavior, such as moving the mole.
- Generating random numbers.
- Using the addition (+) and subtraction (–) blocks.

Getting Started

Connect to the App Inventor website and start a new project. Name it "MoleMash" and also set the screen's title to "MoleMash". Click Connect and connect your device or emulator for live testing.

Download the picture of a mole from *http://appinventor.org/bookFiles/MoleMash/mole.png*. In the Component Designer, in the Media section, click "Upload File," browse to where the file is located on your computer, and then upload it to App Inventor.

Designing the Components

You'll use these components to make MoleMash:

- A `Canvas` that serves as a playing field.
- An `ImageSprite` that displays a picture of a mole and can move around and sense when the mole is touched.
- A `Sound` that vibrates when the mole is touched.
- `Labels` that display "Hits: ", "Misses: ", and the actual numbers of hits and misses.
- `HorizontalArrangements` to correctly position the `Labels`.
- A `Button` to reset the numbers of hits and misses to 0.
- A `Clock` to make the mole move once per second.

Table 3-1 shows all of the components you'll be using.

Table 3-1. The complete list of components for MoleMash

Component type	Palette group	What you'll name it	Purpose
`Canvas`	Drawing and Animation	`Canvas1`	The container for `ImageSprite`.
`ImageSprite`	Drawing and Animation	`Mole`	The user will try to touch this.
`Button`	User Interface	`ResetButton`	The user will press this to reset the score.
`Clock`	User Interface	`Clock1`	Control the mole's movement.
`Sound`	Media	`Sound1`	Vibrate when the mole is touched.
`Label`	User Interface	`HitsLabel`	Display "Hits: ".
`Label`	User Interface	`HitsCountLabel`	Display the number of hits.
`HorizontalArrangement`	Layout	`Horizontal Arrangement1`	Position `HitsLabel` next to `HitsCountLabel`.
`Label`	User Interface	`MissesLabel`	Display "Misses: ".
`Label`	User Interface	`MissesCountLabel`	Display the number of misses.
`HorizontalArrangement`	Layout	`Horizontal Arrangement2`	Position `MissesLabel` next to `MissesCountLabel`.

Placing the Action Components

In this section, you will place the components necessary for the game's action. In the next section, you will place the components for displaying the score.

1. From the Drawing and Animation drawer, drag in a `Canvas` component, leaving it with the default name `Canvas1`. Set its `Width` property to "Fill parent" so that it is as wide as the screen, and set its `Height` to 300 pixels.
2. Again from the Drawing and Animation drawer, drag in an `ImageSprite` component, placing it anywhere on `Canvas1`. At the bottom of the Components list, click Rename and change its name to "Mole". Set its `Picture` property to *mole.PNG*, which you uploaded earlier.
3. From the User Interface drawer, drag in a `Button` component, placing it beneath `Canvas1`. Rename it "ResetButton" and set its `Text` property to "Reset".
4. Also from the User Interface drawer, drag in a `Clock` component. It will appear at the bottom of the Viewer in the "Non-visible components" section.

5. From the Media drawer, drag in a `Sound` component. It, too, will appear in the "Non-visible components" section.

Your screen should now look something like Figure 3-2 (although your mole might be in a different position).

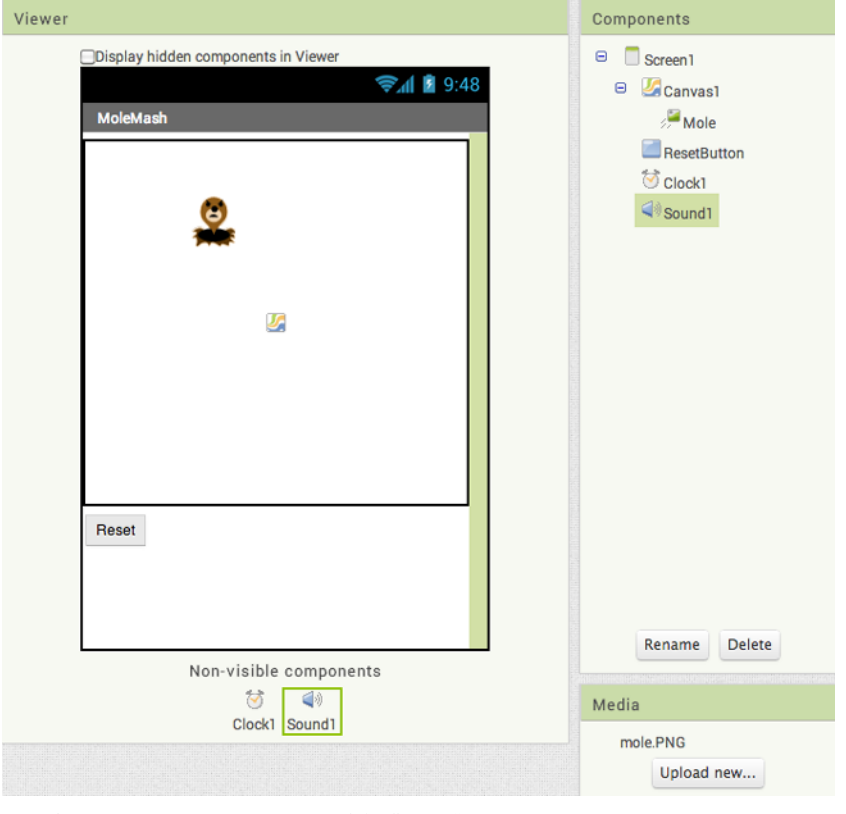


Figure 3-2. The Component Designer view of the "action" components

Placing the Label Components

You will now place components for displaying the user's score—specifically, the number of hits and misses.

1. From the Layout drawer, drag in a `HorizontalArrangement`, placing it beneath the `Button` and keeping the default name of `HorizontalArrangement1`.
2. From the User Interface drawer, drag two `Labels` into `HorizontalArrangement1`.
   - Rename the left `Label` "HitsLabel" and set its `Text` property to "Hits: " (making sure to include a space after the colon).

- Rename the right `Label` "HitsCountLabel" and set its `Text` property to the number 0.

3. Drag in a second `HorizontalArrangement`, placing it beneath `HorizontalArrangement1`.
4. Drag two Labels into `HorizontalArrangement2`.
   - Rename the left `Label` "MissesLabel" and set its `Text` property to "Misses: " (making sure to include a space after the colon).
   - Rename the right `Label` "MissesCountLabel" and set its `Text` property to the number 0.

Your screen should now look like something like Figure 3-3.

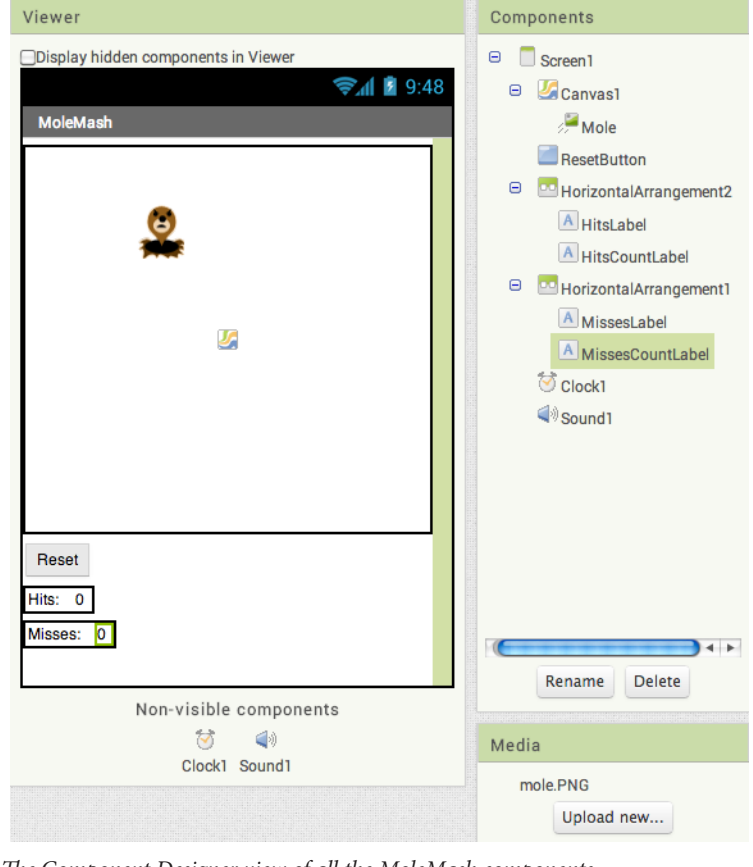


Figure 3-3. The Component Designer view of all the MoleMash components

Adding Behaviors to the Components

After creating the preceding components, let's move to the Blocks Editor to implement the program's behavior. Specifically, we want the mole to move to a random location on the canvas every second. The user's goal is to tap on the mole wherever it appears, and the app will display the number of times the user hits or misses the mole. (Note: We recommend using your finger, not a mallet!) Pressing the Reset button resets the number of hits and misses to 0.

Moving the Mole

In the programs you've written thus far, you've called built-in procedures such as `Vibrate` in HelloPurr. Wouldn't it be nice if App Inventor had a procedure that moved an `ImageSprite` to a random location on the screen? The bad news: it doesn't. The good news: you can create your own procedures! Just like the built-in procedures, your procedure will show up in a drawer and you can use it anywhere in the app.

Specifically, we will create a procedure to move the mole to a random location on the screen, which we will name `MoveMole`. We want to call `MoveMole` at the start of the game, when the user successfully taps the mole, and once per second.

Creating the MoveMole Procedure

To understand how to move the mole, we need to look at how Android graphics work.

The canvas (and the screen) can be thought of as a grid with *x* (horizontal) and *y* (vertical) coordinates, where the (x, y) coordinates of the upper-left corner are (0, 0). The *x* coordinate increases as you move to the right, and the *y* coordinate increases as you move down, as shown in Figure 3-4. The x and y properties of an `ImageSprite` indicate where its upper-left corner is positioned; thus, the mole in the upper-left corner in Figure 3-4 has x and y values of 0.

To determine the maximum available x and y values so that `Mole` fits on the screen, we need to make use of the `Width` and `Height` properties of `Mole` and `Canvas1`. (The mole's `Width` and `Height` properties are the same as the size of the image you uploaded. When you created `Canvas1`, you set its `Height` to 300 pixels and its `Width` to "Fill parent," which copies the width of its parent element, which in this case is the screen.) If the mole is 36 pixels wide and the canvas is 200 pixels wide, the x coordinate of the left side of the mole can be as low as 0 (all the way to the left) or as high as 164 (200 – 36, or `Canvas1.Width` – `Mole.Width`) without the mole extending off the right edge of the screen. Similarly, the y coordinate of the top of the mole can range from 0 to `Canvas1.Height` – `Mole.Height`.

Figure 3-5 shows the procedure you will create, annotated with descriptive comments (which you can optionally add to your procedure).

To randomly place the mole, we will want to select an x coordinate in the range from 0 to *Canvas1.Width – Mole.Width*. Similarly, we will want the y coordinate to be in the range from 0 to *Canvas1.Height – Mole.Height*. We can generate a random number through the built-in procedure *random integer*, which you can find in the Math drawer. You will need to change the default "from" parameter from 1 to 0 and replace the "to" parameters, as shown in Figure 3-5.

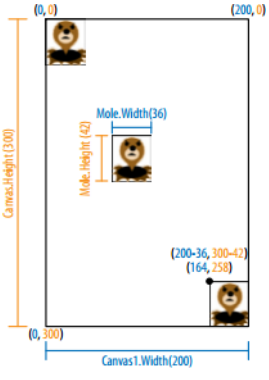


Figure 3-4. Positions of the mole on the screen, with coordinate, height, and width information; x coordinates and widths are shown in blue, while y coordinates and heights are in orange

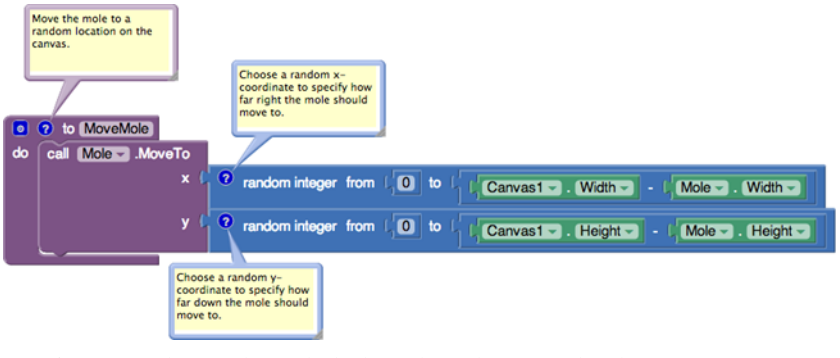


Figure 3-5. The MoveMole procedure, which places the mole in a random location

To create the procedure:

1. In the Blocks Editor, click the Procedure drawer.
2. Drag out the `to procedure` block (containing "do," not "result").
3. On the new block, click the text "procedure" and type "MoveMole" to set the name of the procedure.
4. Because we want to move the mole, click the Mole drawer, and drag `call Mole.MoveTo` into the procedure, to the right of "do." Note the open sockets on the right side which indicate that we need to provide x and y coordinates.

5. To specify that the new x coordinate for the mole should be between 0 and `Canvas1.Width` – `Mole.Width`, as discussed earlier, do the following:
   - From the Math drawer, drag in the `random integer from` block, putting the plug (protrusion) on its left side into the "x" socket on `call Mole.MoveTo`.
   - Change the number "1" block on the "from" socket by clicking it and then entering the number "0".
   - Discard the number "100" by clicking it and pressing your keyboard's Del or Delete key, or by dragging it to the trash can.
   - From the Math drawer, drag in a subtraction (–) block and place it into the "to" socket.
   - From the Canvas1 drawer, select the `Canvas1.Width` block and drag it to the left side of the subtraction operations.
   - Similarly, click the Mole drawer and drag `Mole.Width` into the workspace. Then, plug that into the right side of the subtraction block.
6. Follow a similar procedure to specify that the y coordinate should be a random integer in the range from 0 to `Canvas1.Height` – `Mole.Height`.
7. Check your results against Figure 3-5.

Calling MoveMole When the App Starts

Now that you've written the `MoveMole` procedure, let's make use of it. Because it's so common for programmers to want something to happen when an app starts, there's a block for that very purpose: `Screen1.Initialize`.

1. Click the Screen1 drawer and drag out `Screen1.Initialize`.
2. Click the Procedures drawer, in which you'll see a `call MoveMole` block. (It's pretty cool that you've created a new block, isn't it?!) Drag it out and place it in `Screen1.Initialize`, as shown in Figure 3-6.

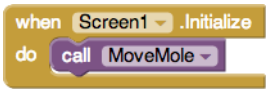


Figure 3-6. Calling the MoveMole procedure when the application starts

Calling MoveMole every Second

Making the mole move every second will require the `Clock` component. We left the `TimerInterval` property for `Clock1` at its default value of 1,000 (milliseconds), or 1

second. That means that every second, whatever is specified in a `Clock1.Timer` block will take place. Here's how to set that up:

1. Click the Clock1 drawer and drag out `Clock1.Timer`.
2. Click the Procedures drawer and drag a `call MoveMole` block into the `Clock1.Timer` block, as shown in Figure 3-7.

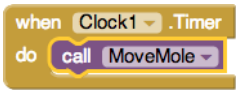


Figure 3-7. Calling the MoveMole procedure when the timer goes off (every second)

If that's too fast or slow for you, you can change the `TimerInterval` property for `Clock1` in the Component Designer to make it move more or less frequently.

Keeping Score

As you might recall, you created two labels, `HitsCountsLabel` and `MissesCountsLabel`, which had initial values of 0. We'd like to increment the numbers in these labels whenever the user successfully taps the mole (a hit) or taps the screen without touching the mole (a miss). To do so, we will use the `Canvas1.Touched` block, which indicates that the canvas was touched, the x and y coordinates of where it was contacted (which we don't need to know), and whether a sprite was tapped (which we do need to know). Figure 3-8 shows the code you will create.

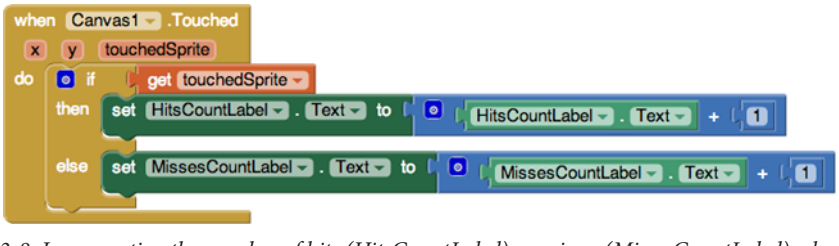


Figure 3-8. Incrementing the number of hits (HitsCountLabel) or misses (MissesCountLabel) when Canvas1 is touched

You can translate the blocks in Figure 3-8 in the following way: whenever the canvas is tapped, check whether a sprite was tapped. Because there's only one sprite in our program, it has to be `Mole1`. If `Mole1` is tapped, add one to the number in `HitsCountLabel.Text`; otherwise, add one to `MissesCountLabel.Text`. (The value of touchedSprite is false if no sprite was touched.)

Here's how to create the blocks:

1. Click the Canvas1 drawer and drag out `Canvas1.Touched`.

2. Click the Control drawer and drag out the `if-then` block. Click its blue icon and add an `else` branch. Then, place it within `Canvas1.Touched`.
3. Mouse over the `touchedSprite` event parameter on `Canvas1.Touched`, and then drag out the `get touchedSprite` block and place it in the test socket of `if-then-else`.
4. Because we want `HitsCountLabel.Text` to be incremented if the test succeeded (if the mole was touched), do the following:
   - From the HitsCountLabel drawer, drag out the `set HitsCountLabel.Text to` block, putting it to the right of "then."
   - Click the Math drawer and drag out a plus sign (+), placing it in the "to" socket.
   - Click the HitsCountLabel drawer and drag the `HitsCountLabel.Text` block to the left of the plus sign.
   - Click the Math drawer and drag a `0` block to the right of the plus sign. Click 0 and change it to 1.
5. Repeat step 4 for `MissesCountLabel` in the `else` section of the `ifelse` block.

Test your app *You can test this new code on your device by tapping the canvas, both on and off the mole, and watching the score change.*

Procedural Abstraction

The ability to name and later call a set of instructions like `MoveMole` is one of the key tools in computer science and is referred to as *procedural abstraction*. It is called "abstraction" because the caller of the procedure (who, in real-world projects, is likely to be different from the author of the procedure) only needs to know what the procedure does (moves the mole), not how it does it (by making two calls to the random-number generator). Without procedural abstraction, big computer programs would not be possible, because they contain too much code for individuals to hold in their head at a time. This is analogous to the division of labor in the real world, where, for example, different engineers design different parts of a car, none of them understanding all of the details, and the driver only has to understand the interface (e.g., pressing the brake pedal to stop the car), not the implementation.

Some advantages of procedural abstraction over copying and pasting code are:

- It is easier to test code if it is neatly segregated from the rest of the program.
- If there's a mistake in the code, it only needs to be fixed in one place.
- To change the implementation, such as ensuring that the mole doesn't move somewhere that it appeared recently, you only need to modify the code in one place.
- Procedures can be collected into a library and used in different programs. (Unfortunately, this functionality is not currently supported in App Inventor.)
- Breaking code into pieces helps you think about and implement the application ("divide and conquer").
- Choosing good names for procedures helps document the code, making it easier for someone else (or you, a month later) to read.

In later chapters, you will learn ways of making procedures even more powerful: adding arguments, providing return values, and having procedures call themselves. For an overview, see Chapter 21.

Resetting the Score

A friend who sees you playing MoleMash will probably want to give it a try, too, so it's good to have a way to reset the number of hits and misses to 0. Depending on which tutorials you've already worked through, you might be able to figure out how to do this without reading the following instructions. Consider giving it a try before reading ahead.

What we need is a `ResetButton.Click` block that sets the values of `HitsCountLabel.Text` and `MissesCountLabel.Text` to 0. Create the blocks shown in Figure 3-9.

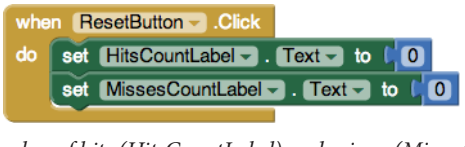


Figure 3-9. Resetting the number of hits (HitsCountLabel) and misses (MissesCountLabel) when the Reset button is pressed

At this point, you probably don't need step-by-step instructions for creating a button click event handler with text labels, but here's a tip to help speed up the process: instead of getting your number from the Math drawer, just type 0, and the block should be created for you. (These kinds of keyboard shortcuts exist for other blocks, too.)

Test your app *Try hitting and missing the mole and then pressing the Reset button.*

Vibrating When the Mole Is Touched

We said earlier that we want the device to vibrate when the user taps the mole, which we can do with the `Sound1.Vibrate` block, as shown in Figure 3-10.

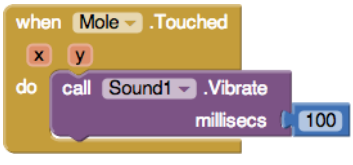


Figure 3-10. Making the device vibrate briefly (for 100 milliseconds) when the mole is touched

Test your app See how the vibration works when you actually tap the mole. If the vibration is too long or too short for your taste, change the number of milliseconds in `Sound1.Vibrate` block.

The Complete App: MoleMash

Figure 3-11 illustrates the blocks for the complete MoleMash app.

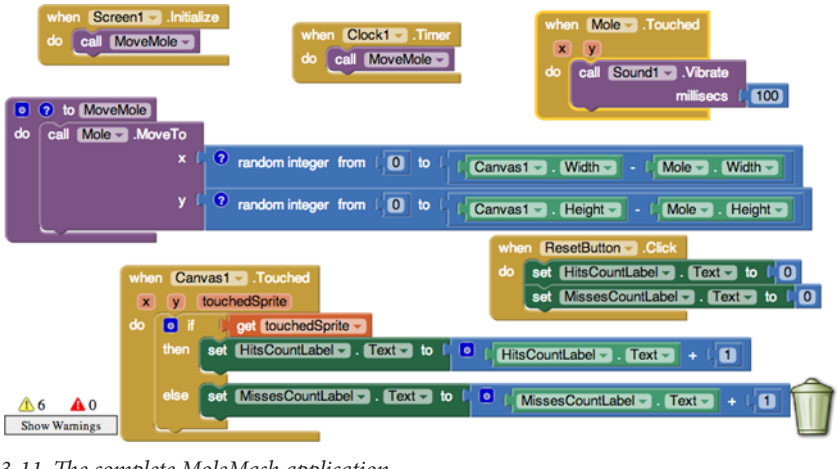


Figure 3-11. The complete MoleMash application

Variations

Here are some ideas for additions to MoleMash:

- Add buttons to let the user make the mole move faster or slower.
- Add a label to keep track of and display the number of times the mole has appeared (moved).
- Add a second `ImageSprite` with a picture of something that the user should *not* hit, such as a flower. If the user touches it, penalize him by reducing his score or ending the game.
- Instead of using a picture of a mole, let the user select a picture with the `Image Picker` component.

Summary

In this chapter, we covered a number of techniques that are useful for apps in general and games in particular:

- The `Canvas` component makes use of an x-y coordinate system, where x represents the horizontal direction (from 0 at the left to `Canvas.Width`–1 at the right), and y the vertical direction (from 0 at the top to `Canvas.Height`–1 at the bottom). The height and width of an `ImageSprite` can be subtracted from the height and width of a `Canvas` to make sure the sprite fits entirely on the `Canvas`.
- You can take advantage of the device's touchscreen through the `Canvas` and `ImageSprite` components' `Touched` methods.
- You can create real-time applications that react not just to user input but also in response to the device's internal timer. Specifically, the `Clock.Timer` block runs at the frequency specified in the *Clock.Interval* property and can be used to move `ImageSprite` (or other) components.
- You can use labels to display scores, which go up (or down) in response to the player's actions.
- You can provide tactile feedback to users through the `Sound.Vibrate` method, which makes the device vibrate for the specified number of milliseconds.
- Instead of just using the built-in methods, you can create procedures to name a set of blocks that can be called just like the built-in ones. This is called procedural abstraction and is a key concept in computer science, enabling code reuse and making complex applications possible.
- You can generate unpredictable behavior with the `random integer` block (in the Math drawer), making a game different every time it is played.

You'll learn more techniques for games, including detecting collisions between moving `ImageSprite` components, in Chapter 5.